Primitive Recursion

Astra Kolomatskaia

21 May 2023

Natural Numbers

A *natural number* is something of the form Z or $\mathsf{S}\ n$ where n stands for any other natural number. For example Z, $\mathsf{S}\ \mathsf{Z}$ and $\mathsf{S}\,(\mathsf{S}\ \mathsf{Z})$ are natural numbers. The intended meaning of these symbols is that Z stands for 0 and S stands for the successor function, which adds 1 to any number. $\mathsf{S}\,(\mathsf{S}\ \mathsf{Z})$ thus denotes the number 2.

Addition, Multiplication, and Factorials

For any two natural numbers n and m, we would like to define another natural number $\mathsf{plus}\ n\ m$. We can define this in the case that m is Z by saying that $\mathsf{plus}\ n\ \mathsf{Z} = n$. In the case that m is of the form $\mathsf{S}\ k$, we can proceed inductively and assume that $\mathsf{plus}\ n\ k$ has already been defined when defining $\mathsf{plus}\ n\ (\mathsf{S}\ k)$. We thus say that $\mathsf{plus}\ n\ \ (\mathsf{S}\ k) = \mathsf{S}\ \ (\mathsf{plus}\ n\ k)$.

We can define multiplication similarly, by saying that $\mathsf{times}\ n\ \mathsf{Z} = \mathsf{Z}$ and that $\mathsf{times}\ n\ \ (\mathsf{S}\ k) = \mathsf{plus}\ \ (\mathsf{times}\ n\ k)\ \ n$. This defines the expression $\mathsf{times}\ n\ m$ for all natural numbers n and m.

Finally, we can define $\mathsf{fact}\ n$. We have that $\mathsf{fact}\ \mathsf{Z} = \mathsf{S}\ \mathsf{Z}$, and that $\mathsf{fact}\ \ (\mathsf{S}\ k) = \mathsf{times}\ \ (\mathsf{fact}\ k)\ \ (\mathsf{S}\ k)$. This case demonstrates that sometimes we need not only knowledge of the value of the recursive case $\mathsf{fact}\ k$, but also of the value of k, when making inductive definitions.

Primitive Recursion

Primitive recursion is a language for defining functions that take natural number inputs and produce a natural number output, using the kind of recipe described above. The only constants in this language are variables, Z, S, and the primitive recursion operator R.

The first thing to clarify is the concept of *variables*. A function of N inputs will simply be defined by giving the body of the function, without any explicit variable bindings. This function body will be understood to live in a context of N variables, which we denote by $v_{N-1}, \ldots, v_0$. In certain cases, we will abstract over new variables, and then the newest variables assume the lowest indices, shifting all of the older indices back. In particular, the freshest variable is always denoted by v_0. This contrasts with an approach in which variables are given explicit names, such as n.

In general, we can define a set of *valid programs* of N inputs, denoted by $\mathsf{Program}_\mathsf{N}$. Z is always a valid program, and if t is a valid program, then $\mathsf{S}\ t$ is a valid program (of the same number of inputs). The variable v_i is a valid program of N inputs just in case $i < \mathsf{N}$ [this condition is known as *well-scopedness*]. Finally, if zc and t are valid program of N inputs, and sc is a valid program of $\mathsf{N}+2$ inputs, then $\mathsf{R}\ zc\ sc\ t$ is a valid program of N inputs. These rules may be summarised as follows:

$$\frac{}{\mathsf{Z} : \mathsf{Program}_\mathsf{N}} \qquad \frac{t : \mathsf{Program}_\mathsf{N}}{\mathsf{S}\ t : \mathsf{Program}_\mathsf{N}} \qquad \frac{i < \mathsf{N}}{v_i : \mathsf{Program}_\mathsf{N}}$$

$$\frac{zc : \mathsf{Program}_\mathsf{N} \qquad sc : \mathsf{Program}_{\mathsf{N}+2} \qquad t : \mathsf{Program}_\mathsf{N}}{\mathsf{R}\ zc\ sc\ t : \mathsf{Program}_\mathsf{N}}$$

Every program defines a natural number, provided that we give values to all of the inputs. A *context* Γ of N inputs is just a list of N numbers. If we fix a length N context Γ, then for every $t : \mathsf{Program}_\mathsf{N}$, we have an interpretation $[\![\ t\ ]\!]_\Gamma : \mathbb{N}$. We can define this as follows: $[\![\ \mathsf{Z}\ ]\!]_\Gamma = 0$, $[\![\ \mathsf{S}\ t\ ]\!]_\Gamma = [\![\ t\ ]\!]_\Gamma + 1$, and $[\![\ v_i\ ]\!]_\Gamma = \Gamma^{v_i}$, where this last expression refers to into the context to pick out a specific number.

The interpretation of R is a bit more subtle. We first define, outside of the language, an operation on natural numbers called natrec. natrec takes as input $zc : \mathbb{N}$, $sc : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$, and $n : \mathbb{N}$, and is defined by induction on n. We have that $\mathsf{natrec}\ zc\ sc\ \mathsf{Z} = zc$ and $\mathsf{natrec}\ zc\ sc\ (\mathsf{S}\ k) = sc\ k\ (\mathsf{natrec}\ zc\ sc\ k)$. Thus zc is the prescribed value at $n = \mathsf{Z}$, and sc specifies how to go from the information of k and the value of the prior recursive call to the value of the successor recursive call. This builds an eliminator, which is then applied to n. Finally, we set $[\![\ \mathsf{R}\ zc\ sc\ n\ ]\!]_\Gamma = \mathsf{natrec}\ [\![\ zc\ ]\!]_\Gamma\ \left(k, r \mapsto [\![\ sc\ ]\!]_{\Gamma,k,r}\right)\ [\![\ n\ ]\!]_\Gamma$. Note that the second argument on the RHS is a function of two natural number arguments.

Examples

Here are the examples from before:

$\mathsf{plus} : \mathsf{Program}_2$
$\mathsf{plus} = \mathsf{R}\ v_1\ (\mathsf{S}\ v_0)\ v_0$

$\mathsf{times} : \mathsf{Program}_2$
$\mathsf{times} = \mathsf{R}\ \mathsf{Z}\ (\mathsf{R}\ v_0\ (\mathsf{S}\ v_0)\ v_3)\ v_0$

$\mathsf{fact} : \mathsf{Program}_1$
$\mathsf{fact} = \mathsf{R}\ (\mathsf{S}\ \mathsf{Z})\ (\mathsf{R}\ \mathsf{Z}\ (\mathsf{R}\ v_0\ (\mathsf{S}\ v_0)\ v_2)\ (\mathsf{S}\ v_1))\ v_0$